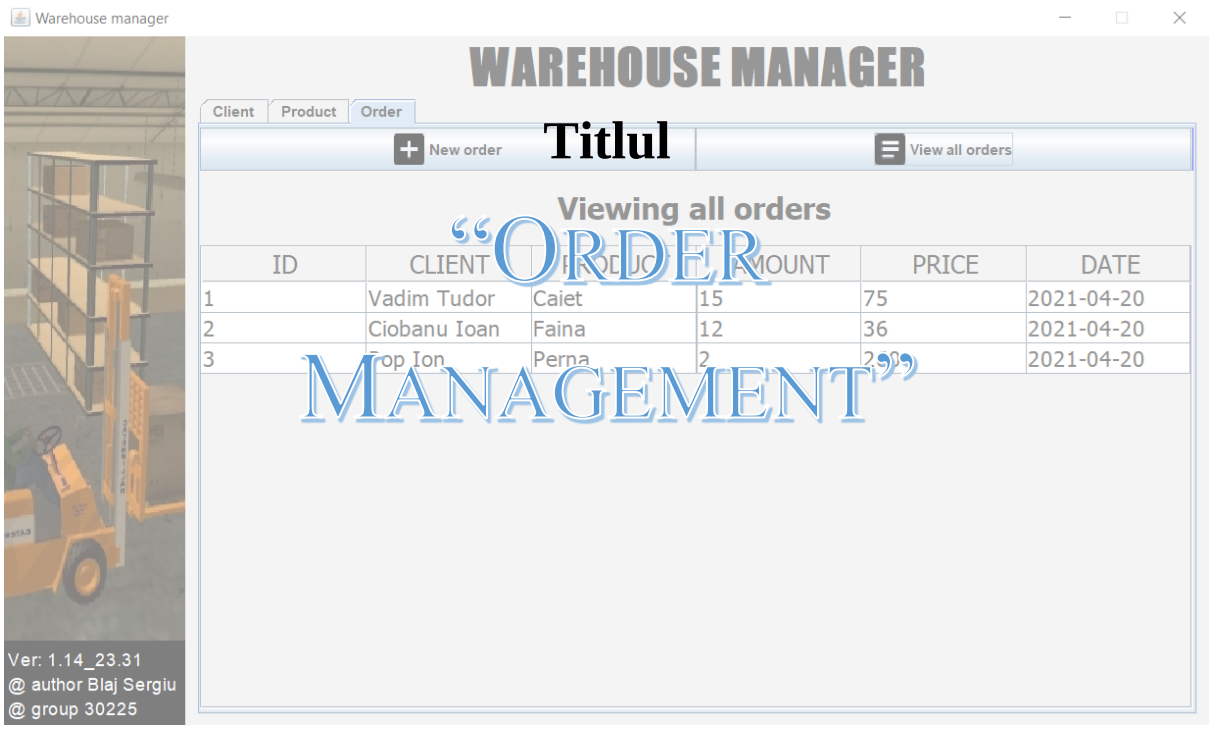
MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**
**DEPARTAMENTUL CALCULATOARE**

# Proiect la disciplina

Tehnici de programare fundamentale




# Titlul "ORDER MANAGEMENT"

## An universitar

2020 – 2021

Blaj Sergiu-Emanuel

An II, grupa 30225

# Cuprins

# 1. Obiectivul temei

Obiectivul principal al temei îl reprezintă proiectarea și implementarea unei aplicații de simulare care procesează comenzile unor clienți. Suplimentar, acest simulator trebuie să asigure:

- utilizarea conceptelor de programare orientată pe obiecte;
- stocarea clineților, a produselor și a comenzilor într-o bază de date relațională;
- dezvoltarea sa folosind un pattern arhitectural;
- dezvoltarea sa folosind minim patru pachete: business logic, data access, presentation și model;
- utilizarea JavaDoc pentru documentarea claselor;
- utilizarea tehnicii de reflexie;
- interacțiunea utilizatorului cu simulatorul printr-o interfață grafică user-friendly, pentru a facilita navigarea prin aplicație: utilizatorul dispune de un panou pentru operații cu clienții, precum crearea, căutarea, editarea sau ștergerea lor, un panou pentru operații cu produsele, precum crearea, căutarea, editarea sau ștergerea lor și un panou pentru operații cu comenzile, precum crearea, căutarea sau afișarea acestora;
- emiterea unui bon fiscal pentru fiecare comandă plasată, în fișierul “bill.txt”;
- testarea și depanarea programului, pentru a verifica funcționalitatea aplicației.


*Figura 1*

# 2. Analiza problemei

## 2.1. Cerințe funcționale

Aplicația dezvoltată trebuie să asigure utilizatorului un panou în care poate efectua operații pe clienți, produse și comenzi. Pentru simplitate, vom considera numai operațiile pe clienți, restul fiind echivalente. Prima operație fundamentală care trebuie să fie făcută este crearea clineților. Utilizatorul are la dispoziție o zonă de text în care va introduce date legate de client, iar la apăsarea butonului „Add”, clientul va fi introdus în baza de date.

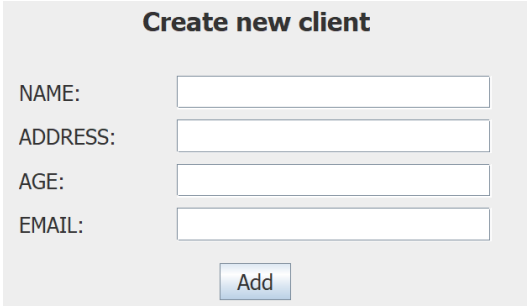



*Figura 2.1.1*

Următoarea operație fundamentală care trebuie să fie făcută este posibilitatea de a edita un client. Mai întâi, într-un panou, utlilizatorul trebuie să introducă id-ul clientului pe care dorește să-l editeze

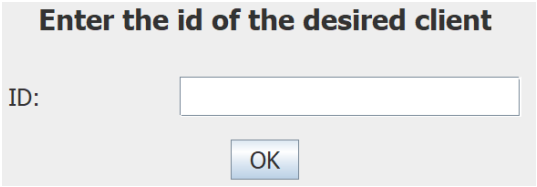



*Figura 2.1.2*

În cazul în care clientul este în baza de date, un panou editabil va conține datele despre client; în caz contrar (clientul nu a fost găsit) va fi afișat un mesaj de alertă corespunzător, tratat în capitolul 5. La apăsarea butonului “Edit”, datele clientului vor fi actualizate în baza de date.

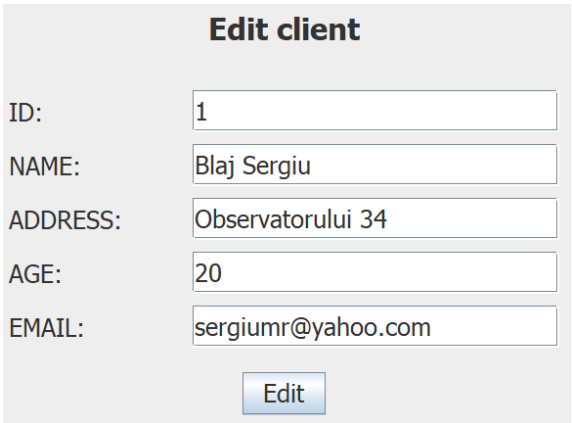



*Figura 2.1.3*

Dacă utilizatorul dorește să șteagă sau să vizualizeze datele unui client, va apărea din nou panoul din figura 2.1.2, pentru introducerea id-ului clientului. Față de editarea unui client, operațiile mai sus menționate nu permit editarea datelor unui client; în plus, la ștergerea unui client, utilizatorul dispune de un buton, care, odată apăsat, va șterge clientul din baza de date.

Ultima operație care poate fi făcută este vizualizarea tuturor clienților într-un tabel:

Viewing all clients

| ID | NAME | ADDRESS | AGE | EMAIL |
|---|---|---|---|---|
| 1 | Blaj Sergiu | Observatorului 34 | 20 | sergiumr@yahoo.com |
| 2 | Pop Ion | Ghiocelul 22 | 21 | pop.ion@yahoo.com |
| 3 | Dumea Mara | Rasboiului 18 | 22 | mara_dumea24@gmail.com |

*Figura 2.1.3*

## 2.2. Cerințe non-funcționale

Pe lângă cerințele de bază, am considerat că următoarele cerințe vor asigura o mai bună interacțiune între calculatorul de polinoame și utilizator. Prima dintre ele, poate chiar cea mai importantă, este ca interfața grafică să fie una intuitivă, astfel ca și cuiva care nu obișnuiește să folosească o aplicație pe calculator să îi fie simplu să o folosească.

În figura 2.2.1 se poate observa că aplicația îndeplinește acest criteriu. În partea stângă, este prezentă o imagine legată de tematica aplicației cât și date despre dezvoltator. În partea principală, apare titlul aplicației, urmat de zonele interactive.

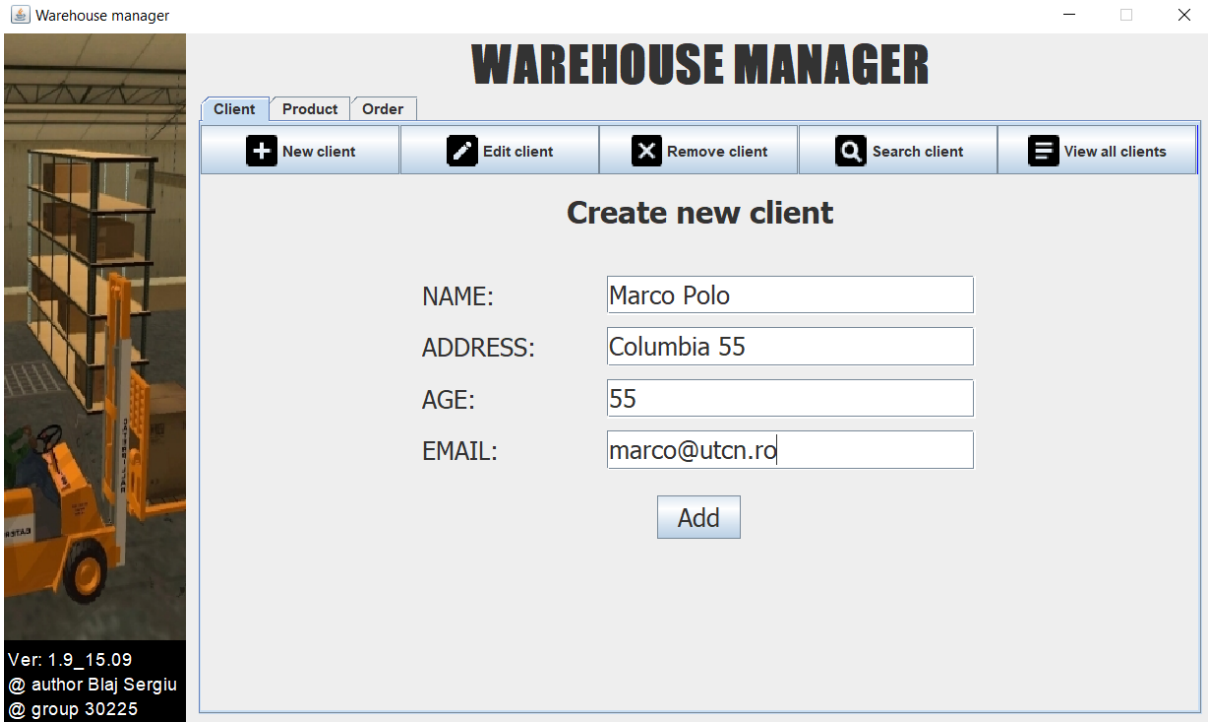



*Figura 2.2.1*

O altă cerință non-funcțională, derivată din cea menționată mai sus, o reprezintă aspectul plăcut al butoanelor și imaginile reprezentative pe care acestea le conțin.

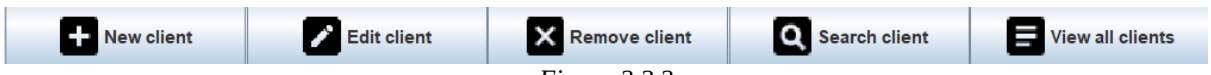



*Figura 2.2.2*

## 2.3. Scenarii de utilizare

Figurile următoare descriu posibilele scenarii de utilizare ale aplicației. Erorile care pot apărea în acest sunt reprezentate de introducerea incorectă a datelor despre client, produse sau comenzi, precum vârsta incorectă, email invalid, sau, în cazul unei comenzi, un id de client sau produs care nu există. Pentru simplitate, vom considera numai operațiile aplicate pe clienți, cele pe produse sau comenzi fiind identice.

Figura 2.3.1 descrie cazul de utilizare al aplicației în care utilizatorul dorește să introducă un nou client în baza de date.

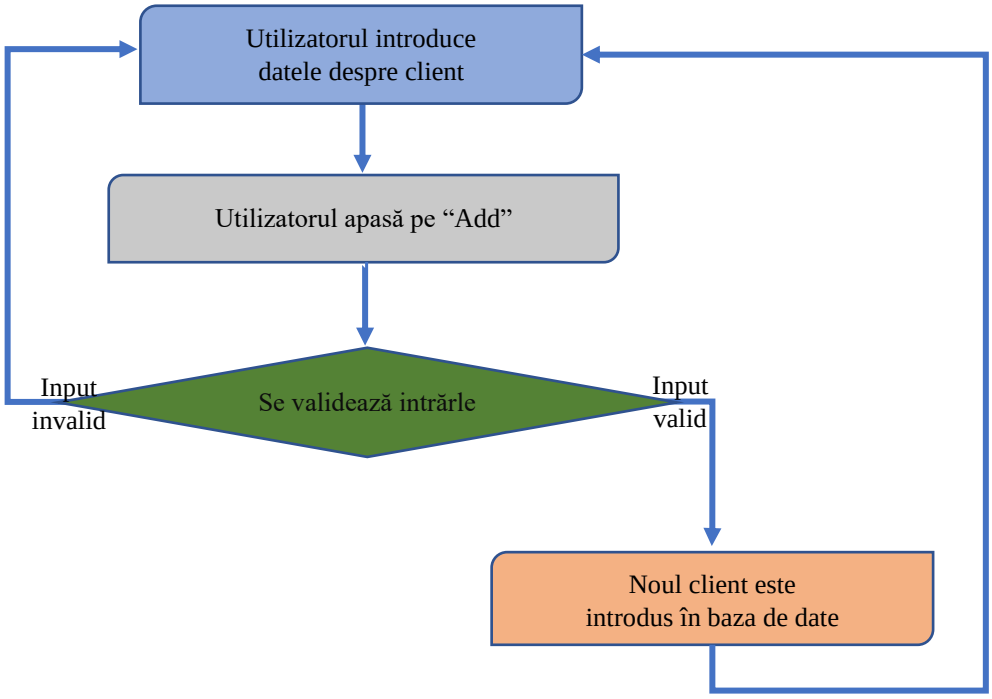



*Figura 2.3.1*

Figura 2.3.2 descrie scenariul de utilizare în cazul în care utilizatorul dorește să editeze, să șteargă sau să caute un client din baza de date.

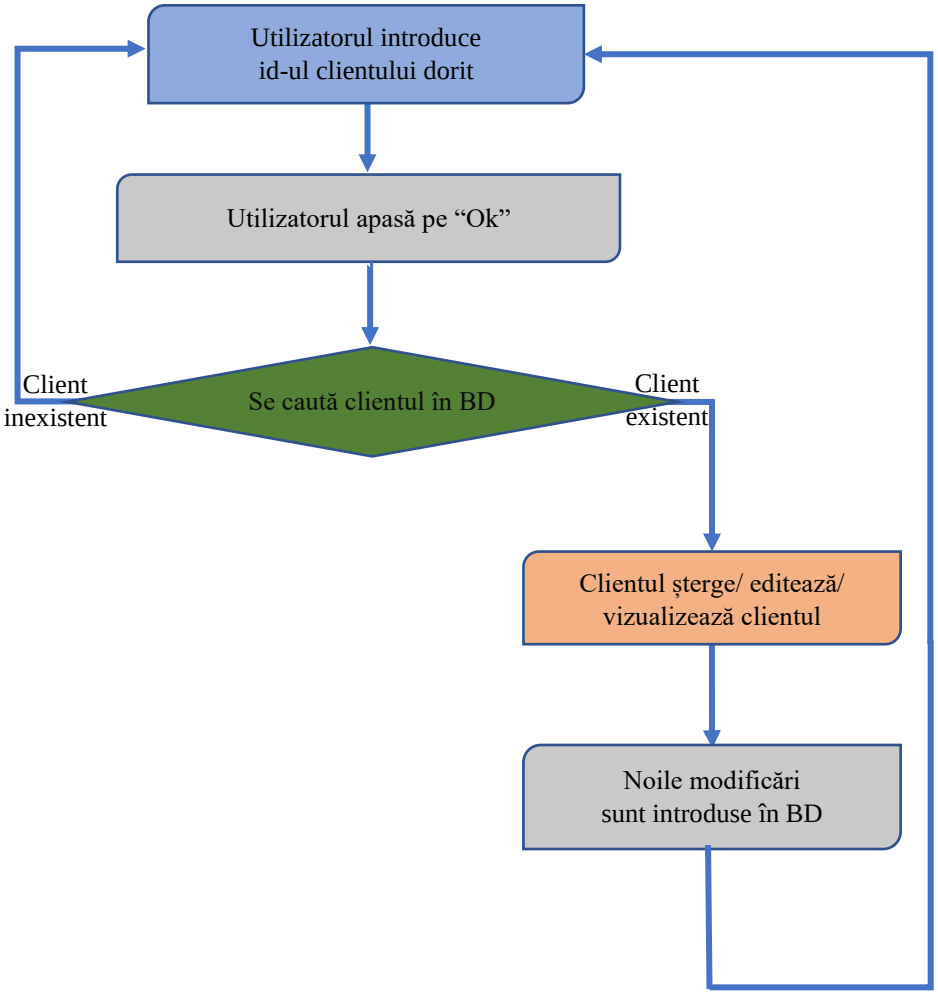



*Figura 2.3.2*

În cazul în care utilizatorul dorește să vizualizeze toți clienții existenți în baza de date, la apăsarea butonului “View all clients”, într-un JTable vor apărea toate intrările din tabelul ‘Clients’.

# 3. Proiectare

## 3.1. Concepte ale programării orientate pe obiect folosite

Aplicația creată a fost implementată în limbajul de programare Java. Fiind dezvoltată într-un limbaj de programare orientat pe obiecte, aplicația modelează obiecte, precum *Client, Product* sau *Order*, utilizează concepte ale programării orientate pe obiect, precum moștenirea, polimorfismul, suprascrierea metodelor, supraîncărcarea metodelor (în special a constructorilor), implementarea de interfețe (în cadrul claselor "ascultători", spre exemplu *CreateObjectListener*, care implementează interfața *ActionListener*, utilă pentru interacțiunea utilizatorului cu interfața grafică.

Deoarece formatul instrucțiunilor SQL asupra manipulării obiectelor de tip *Client, Product* sau *Order* sunt asemănătoare, dar și panourile din interfața grafică referitoare la acestea, a fost folosită tehnica reflexiei. De asemenea, aplicația a fost organizată într-o arhitectură "pe straturi", ce utilizează mai multe pachete, care va fi prezentată în cele ce urmează.

## 3.2. Pachetul Presentation

În figura 3.2 este reprezentată structura pachetului *Presentation*. Aici au fost considerate clasele care alcătuiesc interfața grafică a aplicației, cât și panourile care conțin operațiile pe clienți, produse sau comenzi.

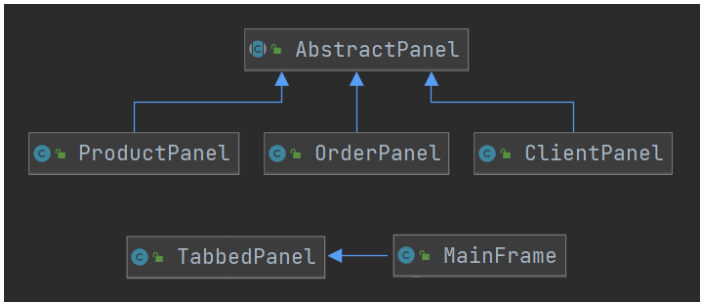



*Figura 3.2*

## 3.3. Pachetul Model

Este pachetul care conține cele trei clase care sunt mapate în baza de date, și anume: *Clienti, Product* și *Order*. Se observă că între client și comandă, dar și între produs și comandă avem o relație de "one-to-many".

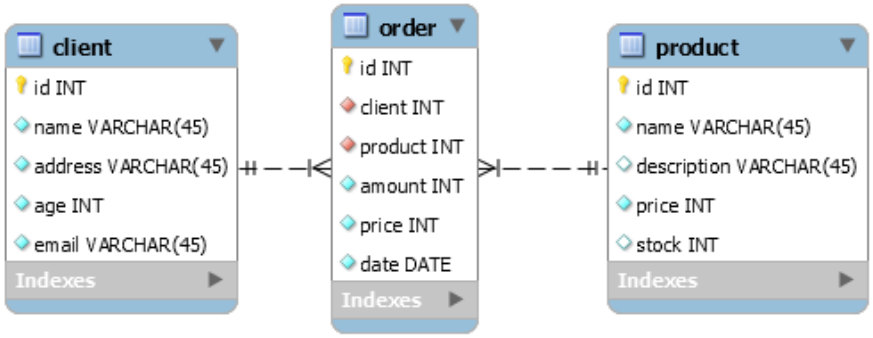



*Figura 3.3*

### 3.4. Pachetul Business Logic

Este pachetul care conține logica aplicației, și anume clasele care fac apeluri către pachetul *Data Access* pentru a prelua informații din baza de date dar și clasele care asigură interacțiunea dintre utilizator cu aplicația.

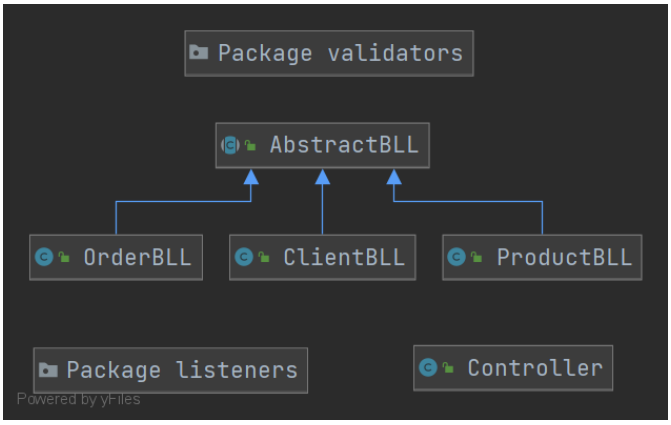



*Figura 3.4*

### 3.5. Pachetul Data Access

În acest pachet sunt conținute clasele care efectuează interogări asupra bazei de date și returnează entități de tipul obiectelor prezente în pachetul *Model*.

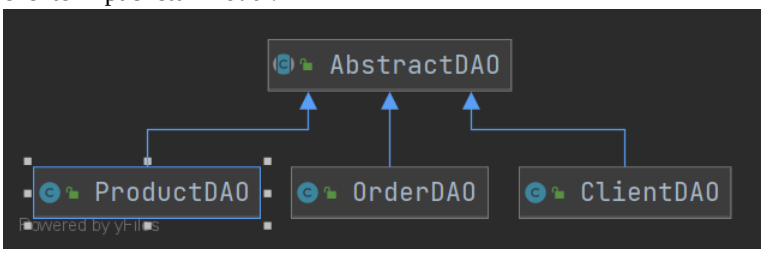



*Figura 3.5*

# 4. Implementare

În cele ce urmează, vor fi descrise clasele folosite de aplicație și modul lor de funcționare. Pentru o mai bună organizare, acestea vor fi descrise în cadrul secțiunii pachetului din care acestea fac parte.

## 4.1. Pachetul Presentation

Pachetul conține clasele care alcătuiesc interfața grafică a aplicației; panourile care conțin operațiile pe clienți, produse sau comenzi.

Clasa *MainFrame* este responsabilă pentru afișarea interfeței grafice. Clasa conține constante String și int, pentru a evita folosirea "numerelor magice" în program, apoi o listă de panouri și butoane. Construcția ferestrei a fost împărțită în mai mulți pași, fiecare pas fiind dictat de constructorul clasei. Spre exemplu, *void setUpFrame()* este responsabilă de a pregăti JFrame-ul (fereastra aplicației), setând dimensiunea ei, titlul ș.a.m.d., astfel că fiecare funcție face ceea ce conține în nume.

Sunt adăugate, rând pe rând, partea din stânga a interfeței, cea care conține imaginea legată de tematica aplicației, informații despre autor, titlul aplicației precum și altele legate de simulare.

Urmează partea interesantă. Observând că fiecare zonă corespunzătoare efectuării operațiilor pe obiectele manipulate de această simulare conține aceleași informații, am creeat clasa *AbstractPanel*, care extinde clasa JPanel și conține o listă de Jlabel-uri și de JTextField-uri care.

Cu ajutorul reflexiei, textul afișat în JLabel este modificat pentru a afișa informația corespunzătoare (concret, în cazul unui client vor fi afișate numele, adresa, vârsta și emailul, pentru un produs numele, descrierea, prețul și stocul ș.a.m.d.).

Cu ajutorul diferitelor metode, se creează iluzia existenței mai multor paneluri interioare: În funcție de operația selectată, șterge anumite texte din panou și le afișează pe altele (de exemplu, metoda *void showCreateObjectPanel()* va afișa aproape toate inputurile legate de un client, în afară de inputul pentru id, deoarece acesta va fi calculat automat la introducerea unui client în baza de date).

O altă caracteristică importantă a acestei clase o reprezintă modul în care sunt afișate obiectele din baza de date în tabele. Cu ajutorul reflexiei, sunt preluate câmpurile care descriu un obiect mapat în baza de date și sunt setate numele coloanelor.

Alte două metode importante le reprezintă *T getFields()*, care preia informația din câmpurile de input puse la dispoziție utilizatorului și returnează un obiect de tipul T, unde T va lua locul lui *Client, Product* sau *Order* și metoda *void setFields(T object)* care primește un obiect de tipul T și va pune informațiile legate de acesta în interfața grafică, utilă în cazul afișării informațiilor despre obiect, editarea acestuia ș.a.m.d.

În final, pe lângă getteri și setteri, sunt prezente metodele care vor fi folosite de *Controller* pentru a adăuga funcționalitate butoanelor.

## 4.2. Pachetul Model

Pachetul conține cele trei clase mapate în baza de date, și anume *Client, Product* și *Order*. Fiecare este definită de variabile instanță specifice obiectului, precum numele său, vâârsta (în cazul unui client), prețul (în cauzl unui produs) sau data (în cazul unei comenzi), diferiți constructori, care vor fi manipulați de reflexie în pachetul următor prezentat, getteri și setteri.

## 4.3. Pachetul Business Logic

Pachetul conține clasele care asigură logica aplicației. Printre acestea, menționăm clasele care modifică obiectele din baza de date. Aceastea primesc obiecte create de clasele din pachetul *Presentation* cu ajutorul *Controllerului*. Înainte de a manipula baza de date, aceste obiecte trebuie să fie validate, fiecare în funcție de tipul său. În cazul unui client, trebuie să ne asigurăm că vârsta acestuia se încadrează în normele legale, sau că adresa

lui de email respectă un anumit model. Sau, o comandă trebuie să fie făcută de un client existent, și numai dacă există suficiente produse de acel tip. Dacă validarea are loc cu succes, aceste clase fac apelul final către clasele și metodele care interoghează baza de date.

Tot în acest pachet au fost definite clasele ce vor asigura interacțiunea utilizatorului cu programul. Clasa principala, *Controller*, asignează fiecărui buton din interfața grafică, o utilizare. Așadar, fiecare buton din program are câte o clasă ce corespunde unui "ascultător" diferit (de exemplu, *CreateObjectListener* este adăugat butonului de adăugare, care, odată apăsat, va prelua valorile din zonele dedicate introducerii acestora, le va valida, iar în caz că acestea sunt corecte, va introduce obiectul în baza de date).

## 4.4. Pachetul Data Access

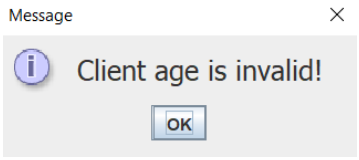
Pachetul conține clasele care vor manipula efectiv intrările mapate în tabelele din baza dedate. În clasa *AbstractDAO* au fost definite, cu ajutorul reflexiei, metodele CRUD, valabile pentru orice obiect. Pentru simplitate, și pentru că restul metodelor funcționează pe același principiu, ne vom opri atenția asupra metodei *int insert(T object)*. Aceasta primește un obiect generic, care este preluat din câmpurile din interfața grafică și-l introduce în baza de date, returnând id-ul intrării corespunzătoare în tabel. Cu ajutorul metodei *String createInsertQuery()* se formează interogarea pentru a insera un obiect în baza de date. Mai apoi, acest string va fi completat cu valorile din obiectul cu pricina și se va efectua propriu-zis inserarea acestuia în baza de date.

O metodă interesantă din această clasă o reprezintă metoda *ArrayList<T> createObjects(ResultSet resultSet)*, care primește un set de rezultate care a fost obținut în urma unei interogări asupra bazei de date și, dintr-o listă de intrări din baza de date returnează o listă de obiecte pe care Java le poate manipula. Iterând, cu ajutorul reflexiei prin câmpurile obiectului, se setează, cu ajutorul setterilor valorile acelui obiect, după care este adăugata în lista finală de obiecte.

Clasele *ClientDAO* și *OrderDAO* extind clasa *AbstractDAO*, însă clasa *ProductDAO*, pe lângă acest lucru, mai folosește alte două metode utilizate în cadrul creării unei noi comenzi, *void decrementStock(int productId, int usedAmount)* și *int getPrice(int productId)*.

# 5. Rezultate

În cele ce urmează vom trece în revistă câteva dintre mesajele de alertă pentru a demonstra corectitudinea aplicației. Pentru simplitate, vom considera erorile sau mesajele care vor apărea în cazul unui client. La crearea unui client, în cazul în care utilizatorul dorește să creeze un client minor, sau un client fără adresă de email, vor apărea erorile de mai jos:



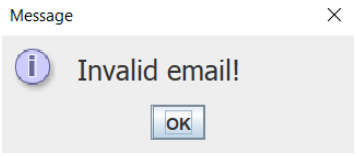




*Figura 5.1*

În caz contrar, câmpurile se validează cu succes iar noul client va fi inserat în baza de date:

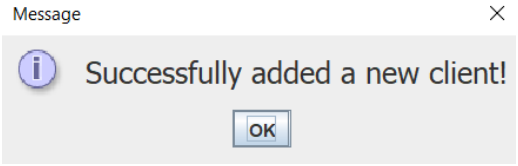



*Figura 5.2*

Dacă se dorește editarea, ștergerea sau găsirea unui client, mai întâi trebuie să se introducă id-ul acestuia. În cazult în care clientul dorit nu există, un mesaj de eroare va semnala acest lucru:

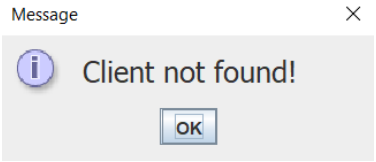



*Figura 5.2*

În caz contrar, dacă clientul cu id-ul specificat este găsit și se dorește ștergerea acestuia din baza de date, un dialog va aștepta confirmarea utilizatorului de a șterge sau nu clientul din baza de date, în cazul în care butonul de remove a fost apăsat din greșeală:

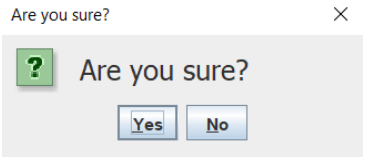



*Figura 5.3*

Presupunând ca la crearea unei noi comenzi se introduce un id de client care nu există în baza de date, un mesaj corespunzător va semnala acest lucru:

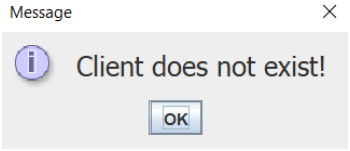



*Figura 5.4*

# 6. Concluzii

În aproape fiecare afacere este nevoie ca beneficiarii activității prestate de un antreprenor, clienții, să fie menținuți într-o bază de date, pentru eventuale prelucrări sau în scopuri de marketing. De asemenea, o afacere care oferă produse destinate vânzării trebuie să țină evidența acestora, iar cer mai simplu ar fi păstrarea lor într-o bază de date relațională. Astfel, în cazul în care un client dorește să achiziționeze un anumit produs, va creea o comandă, care va fi inserată în baza de date și care va ține anumite detalii, precum clientul care a plasat comanda, produsul comandat, data plasării comenzii, cantitatea comandată, prețul comenzii etc.

În urma acestei teme, consider că mi-am îmbunătățit stilul de programare, respectând convențiile de numire Java sau de organizare a codului în pachete și subpachete. De asemenea, mi-am consolidat cunoștințele deja existente legate despre programarea orientată pe obiecte, precum moștenirea, polimorfismul sau elementele de interfață grafică din pachetul Java.swing dar și tipurile generice, punând bazele unor cunoștințe noi de Java, cum ar fi tehnica reflexiei.

În momentul de față, aplicația permite crearea comenzilor cu un produs, însă, în viitor, aceasta ar putea fi dezvoltată astfel încât să permită introducerea mai multor produse în aceeași comandă. De asemenea, ar putea fi dezvoltată o versiune în care produsele vor avea imagini sugestive, nu doar nume și descirere. La fel s-ar putea proceda și pentru clienți, fiecare având posibilitatea de a avea un avatar.

O altă posibilă dezvoltare ulterioară ar putea permite căutarea unui client după nume, adresă sau email, la fel și în cazul unui produs sau, mai important, a unei comenzi. Luând ca exemplu lumea reală, fiecare client ar putea avea o anumită sumă de bani, care, asemenea stocului produselor în cazul plasării unor noi comenzi, se decrementează, la fel s-ar putea întâmpla și cu această sumă de bani. În cazul unui client care plasează comenzi regulat, sau de o valoare mai mare, acesta ar putea primi un anumit discount.

# 7. Bibliografie

https://www.oreilly.com/library/view/building-java-enterprise/0596001231/ch02s03.html

https://www.javatpoint.com/steps-to-connect-to-the-database-in-java

https://www.oracle.com/technical-resources/articles/java/javareflection.html

https://www.geeksforgeeks.org/reflection-in-java/

https://docs.oracle.com/javase/tutorial/reflect/index.html

https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html

https://www.baeldung.com/javadoc

https://www.oracle.com/ro/technical-resources/articles/java/javadoc-tool.html